CS189A - Capstone

Chandra Krintz
Department of Computer Science
UC Santa Barbara
http://www.cs.ucsb.edu/~ckrintz/
The software requirements document

• The software requirements document is the official statement of what is required of the system developers.
• Should include both a definition of user requirements and a specification of the system requirements.
• Defines **WHAT the system should, not HOW it should do it.**
  – Design comes later
  – A living document that **evolves as you prototype** (the Agile SWE way)
Agile Requirements Specification

1. Define project specifics
2. Team goals and objectives
3. Background and strategic fit
4. Assumptions
5. User Stories or Use Cases
6. User Interaction and Design
7. Questions
8. What we’re NOT Doing

• Evolve the document over time, concurrently with development
PRDv1: Your Living Requirements Document: A Shared Google Doc

• Authors, Team, Project Title
• Intro – including problem, innovation, science, core technical advance (2-3 pages)
  – Define project specifics, team goals/objectives, background, and assumptions
• System architecture overview
  – High level diagram (1 page)
  – User interaction and design (1+ page)
• Requirements (functional and non-functional)
  – User stories or use cases (links) → 10 for PRDv1 prioritized
  – Prototyping code, tests, metrics (5+ user stories): github commits/issues
• System models: contexts, sequences, behavioral/UML, state
• Appendices
  – Technologies employed
PRDv2: Your Living Requirements Document: A Shared Google Doc

• Authors, Team, Project Title
• Intro: problem, innovation, science, core technical advance
  – Define project specifics, team goals/objectives, background, and assumptions
• System architecture overview
  – High level diagram (1 page)
  – User interaction and design (1 page)
• Requirements (functional and non-functional)
  – User stories or use cases (links) → 20+ for PRDv2 prioritized
  – Prototyping code, tests, metrics (10+ user stories): github commits/issues
• System models (1+ pages)
  – Contexts, interactions, structural, behavioral (UML)
  – Use cases, sequencing, event response, system state, classes/objects
• Appendices - Technologies employed
The software requirements document

• The software requirements document is the official statement of what is required of the system developers.
• Should include both a definition of user requirements and a specification of the system requirements.
• Defines WHAT the system should, not HOW it should do it.
  – Design comes later
• A living document that evolves as you prototype (the Agile SWE way)

• Agile and extreme SWE processes express requirements as
  – Use cases – how a system will act
  – Or as scenarios called user stories (describe result/benefit of thing)
Use Cases

• Use cases document the behavior of the system from users’ point of view.
  – By user we mean anything external to the system
    actors – scope – goals – steps – success

• An actor is a role played by an outside entity that interacts directly with the system
  – An actor can be a human, or a machine or program
  – Actors are shown as stick figures in use case diagrams

Customer
Use Cases

• A **use case** describes the possible **sequences of interactions** among the system and one or more actors in response to some initial stimulus by one of the actors
  – Each way of using the system is called a use case
  – A use case is not a single scenario but rather a description of a **set of scenarios**
  – For example: *Creating an account*
  – Individual use cases are shown as named ovals in use case diagrams

  ![Creating an account](image)

• A use case involves a sequence of interactions between the initiator and the system, possibly involving other actors.

• In a use case, the system is considered a black-box.

**We are only interested in describing externally visible behavior**

**Have preconditions and postconditions**
Use Cases

• To define a use case, group all transactions that are similar in nature

• A typical use case might include a main case, with alternatives taken in various combinations and including all possible exceptions that can arise in handling them

• Description of a use case should include events exchanged between objects and the operations performed by the system that are visible to actors

• Have preconditions and postconditions
  - Precondition states all assumptions about state/environment of system that impacts the actor(s) in this use case
  - Postcondition is an acceptance test (how to know when implementation is complete) and describes externally visible state/environmental changes
Use Cases

• To define a use case, group all transactions that are similar in nature
• A typical use case might include a main case, with alternatives taken in various combinations and including all possible exceptions that can arise in handling them
  – Use case for a bank: Performing a Transaction at the Counter
    • Subcases could include Making Deposits, Making Withdrawals, etc., together with exceptions such as Overdrawn or Account Closed
  – Apply for a Loan could be a separate use case since it is likely to involve very different interactions
• Description of a use case should include events exchanged between objects and the operations performed by the system that are visible to actors
• Have preconditions and postconditions
  – Precondition states all assumptions about state/environment of system that impacts this use case
  – Postcondition as an acceptance test (how to know when implementation is complete)
Generalization in Use Case Diagrams

UC Santa Barbara

Customer
- Individual Customer
- Corporate Customer

Validate User
- Check Password
- Retinal Scan

Indicates generalization
Use case: Update Benefits


Precondition: Employee has logged on to the system and selected “update benefits” option

Flow of Events:

Basic Path:
1. System retrieves employee account from Employee Account Database
2. System asks employee to select medical plan type; uses Update Medical Plan
3. System asks employee to select dental plan type; uses Update Dental Plan
   ...

Alternative Paths:
If health plan is not available in the Employee’s area the employee is informed and asked to select another plan (exceptional cases that must be handled)
   Employee selects cancel, logs out, or leaves page at any point prior to confirming the update

Postcondition: Employee account plan type has been updated in the Employee Account Database

Note that code tests can be written for pre/post conditions
User Stories

• Similar to Use Cases but not the same
  – User stories are centered on the result and the benefit of the thing you’re describing, whereas use cases are more granular, and describe how your system will act. From: http://www.boost.co.nz/blog/2012/01/use-cases-or-user-stories/

• Use cases: actors – scope – goals – steps – success
  – Details of most important requirements worked out ahead of time to ensure that everyone is on the same page
  – Useful for groups of similar stories and describing overall system
    • Use cases decompose stories into actions in the system

• User stories: scope of a feature + acceptance criteria
  – Each feature is captured as a story; stories easily prioritized
  – A story is a place holder for discussion and planning poker in a sprint
User Stories

• Stems from Behavior Driven Development (BDD)
  – Employed in XP/Agile processes
  – Improves communication/understanding of requirements by all involved

• An outside-in methodology
  – Encourage discovery: drill down on a feature set to achieve desired (business) outcomes

• See for examples
  – Dan North: “What’s in a Story?”
  – Agile Modeling: “Introduction to User Stories”
Writing Good User Stories

- It's typically difficult to get started writing good user stories
  - Here are 4 steps to make it easier

1. As a [role], I can [feature] so that [reason]
2. Use index cards and a sharpie
3. Make it testable with acceptance criteria or demo plan
4. Connect the dots

From: http://codesqueeze.com/the-easy-way-to-writing-good-user-stories/
As a [role], I can [feature] so that [reason]

- Role – a person; feature – something your project does; reason – a solution to a problem the person has
  - This is a pattern that is commonly used for stories

  As a account owner, I can check my balance online so that I can keep a daily balance 24 hours a day.

- Variations
  - As a [role], I want [feature] because [reason]
  - As a [role], I can [feature]
  - As a [role], I can [feature] so that [reason]
Use index cards and a sharpie

- Although there is software out there to help you with this
  - Jira, Trello, Pivotal tracker

- Physically writing out stories facilitates keeping the story clear, concise, and of the appropriate size
  - Keep them short and sweet and unambiguous
    - Goal is to aid communication, not overly detailed or long-winded
  - It also enables you to doodle/draw the outline of the user interface

- If it doesn’t fit, break up the story into sub-stories
Make it testable with acceptance test or demo

• If they are short and sweet and without detail, how do we know when they are “done”?

• Include an acceptance test (what to demo when done):

  Scenario 1: Title
  Given [context]
  And [some more context]…
  When [event]
  Then [outcome]
  And [another outcome]…

  Example
  Scenario 1: Account balance is negative
  Given the account’s balance is below 0
  And there is not a scheduled direct deposit that day
  When the account owner attempts to withdraw money
  Then the bank will deny it
  And send the account owner a nasty letter.

• All tests should fit on back of story card (in sharpie)
  – If they don’t break up the story into two
  – You should be able to code them in a few lines of code